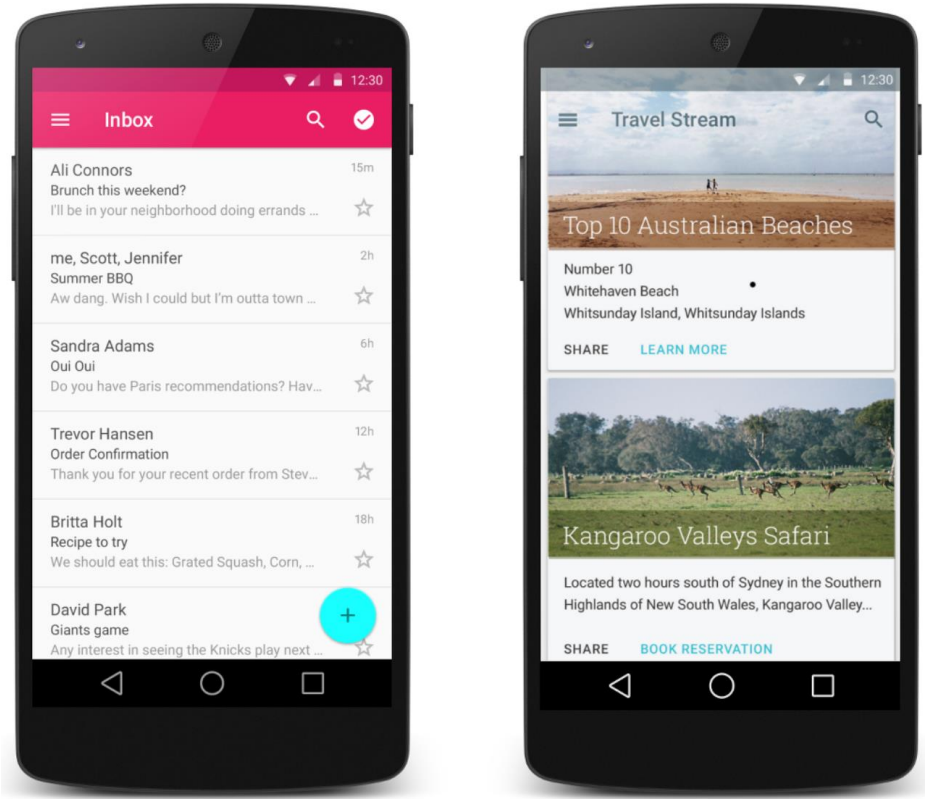


Displaying and Interacting with Lists

Dr. Abdelkarim Erradi

CSE@QU

Outline

1. Displaying a List
2. Interacting with a List

Displaying a List

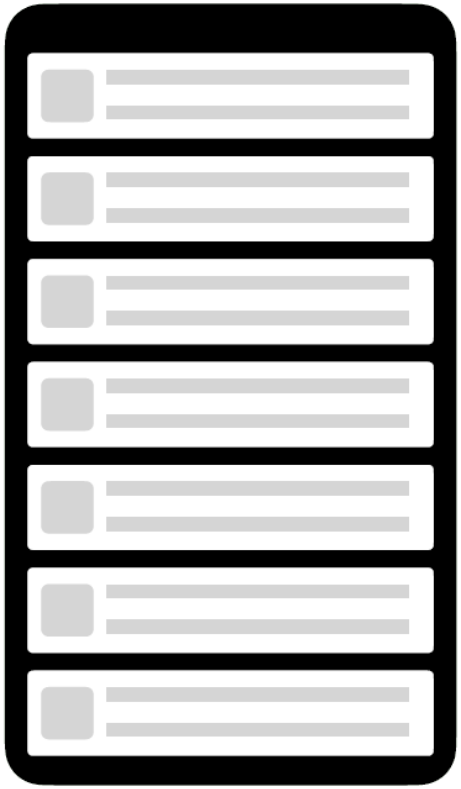
Back

Displaying a List

- In apps it is common to display collections of items
- For displaying a small collection of items, a **`Colum`** or **`Row`** layouts could be used
  - The **`verticalScroll()`** modifier could be applied to make the `Column` scrollable
  - The **`horizontalScroll()`** modifier could be applied to make the `Row` scrollable
- For displaying a large list, using a Column/Row layout can cause performance issues
  - Since all the items will be composed and laid out whether or not they are visible
  - Use a Lazy List (i.e., LazyColumn or LazyRow) to only compose and lay out items which are **visible on screen**

Displaying a List

Making the `Column` scrollable by using the `verticalScroll()` modifier



```
@Composable
fun SurahsList(surahs: List<Surah>) {
    Column(modifier =
     Modifier.verticalScroll(rememberScrollState())
    ) {
        if (surahs.isEmpty()) {
            Text("Loading surahs failed.")
        } else {
            surahs.forEach {
                SurahCard(surah = it)
            }
        }
    }
}
```

Common Modifiers

- `Column(modifier = Modifier.verticalScroll(rememberScrollState())`

Makes the column scrollable

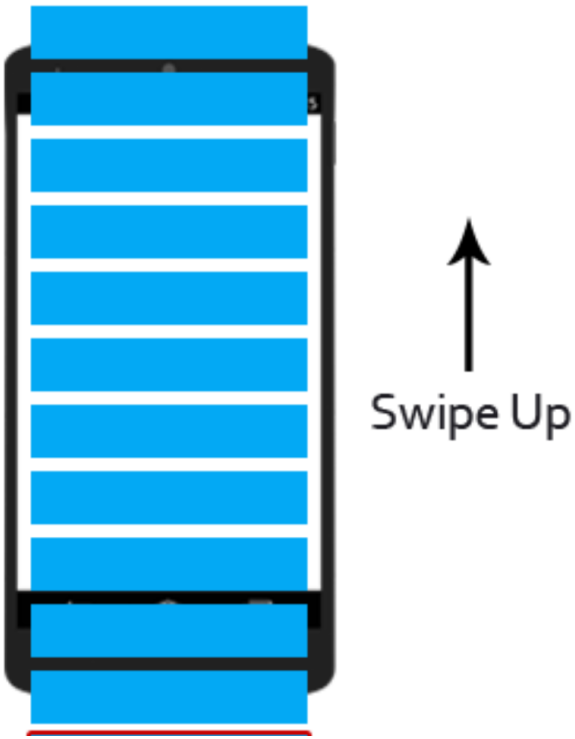
- `Row(modifier = Modifier.horizontalScroll(rememberScrollState())`

Makes the row scrollable

- `Modifier.fillMaxWidth() / .fillMaxHeight() / .fillMaxSize()` - occupy the available space

What is a Lazy List?

- A Lazy List is a scrollable container for displaying a list of composables
  - **LazyColumn** produces a vertically scrolling list, and **LazyRow** produces a horizontally scrolling list
- A flexible container for efficiently displaying, and interacting with large sets of data
  - As user scrolls, views are created to display new items
  - Efficient as it uses a limited number of views



Lazy List methods

- Lazy List provides several functions for describing items in the layout:
  - `item()` to add a single item (e.g., header/footer)
  - `items(aList)` to add multiple items
  - `itemsIndexed(aList)` to add multiple items and provides an index

```
import androidx.compose.foundation.lazy.items
…
LazyColumn {
    items(surahs) {
        SurahCard(it)
    }
}
```

Styling a List - Content spacing

- Use `Arrangement.spacedBy()` to add spacing in-between items

```
LazyColumn(
  verticalArrangement = Arrangement.spacedBy(4.dp),
) {}
```

- Similarly, for LazyRow:

```
LazyRow(
  horizontalArrangement = Arrangement.spacedBy(4.dp),
) {}
```

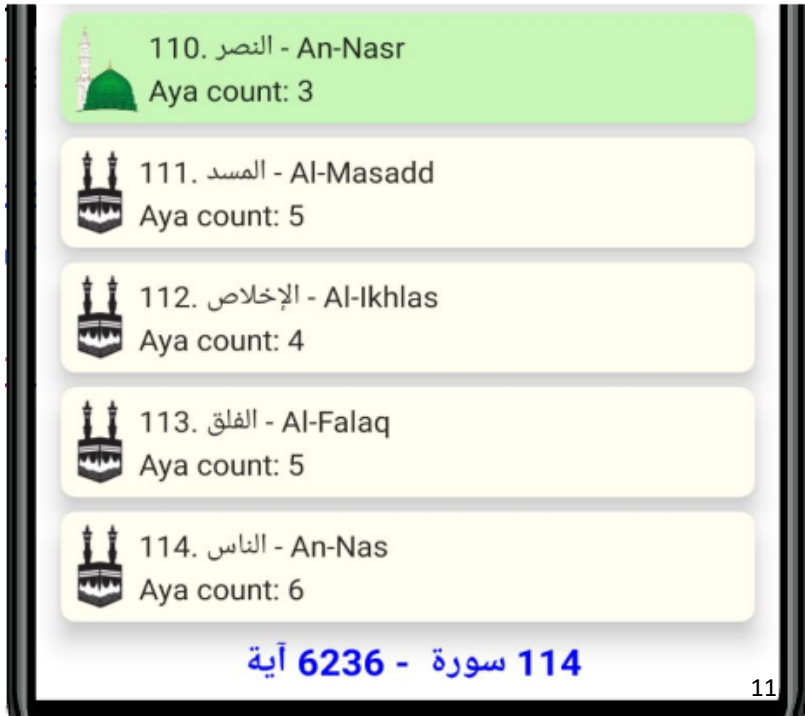
Styling a List – Content padding

- To add padding around the edges of the content pass some **`PaddingValues`** to the **`contentPadding`** parameter

```
LazyColumn(
    contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp)
) {}
```

  - Adds 16.dp of padding to the horizontal edges (left and right), and then 8.dp to the vertical edges (top and bottom) of the content
  - Padding is applied to the content, not to the LazyColumn itself

```
LazyColumn(contentPadding =
    PaddingValues(horizontal = 8.dp, vertical = 8.dp),
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    item {
        Text(
            text = "سور القرآن الكريم",
            textAlign = TextAlign.Center,
            modifier = Modifier.fillMaxWidth(),
            style = TextStyle(
                fontWeight = FontWeight.Bold,
                fontSize = 24.sp,
                color = Color.Blue
                textDirection = TextDirection.Rtl
            )
        )
    }
    items(surahs) {
        SurahCard(it)
    }
    item {
        Text(
            text = "$surahCount سورة - $ayaCount آية",
            textAlign = TextAlign.Center,
            modifier = Modifier.fillMaxWidth(),
            style = TextStyle(
                fontWeight = FontWeight.Bold,
                fontSize = 20.sp,
                color = Color.Blue,
                textDirection = TextDirection.Rtl
            )
        )
    }
}
```



Interacting with a List

Back

Search

Sort

Handling Item Clicked Event

Add/Update a list item

Swipe to Delete and Undo

Pull to refresh

Sticky headers

Summary

- Use the `verticalScroll` or `horizontalScroll` modifiers to display a small list of composables in a Column or a Row
- For dynamic and larger lists use `LazyColumn` and `LazyRow` for the vertical and horizontal scenarios, respectively
- You can program various interactions with a displayed list including *search*, *sort*, *refresh*, *add*, *update* and *delete*

Resources

- Displaying a list using Jetpack Compose

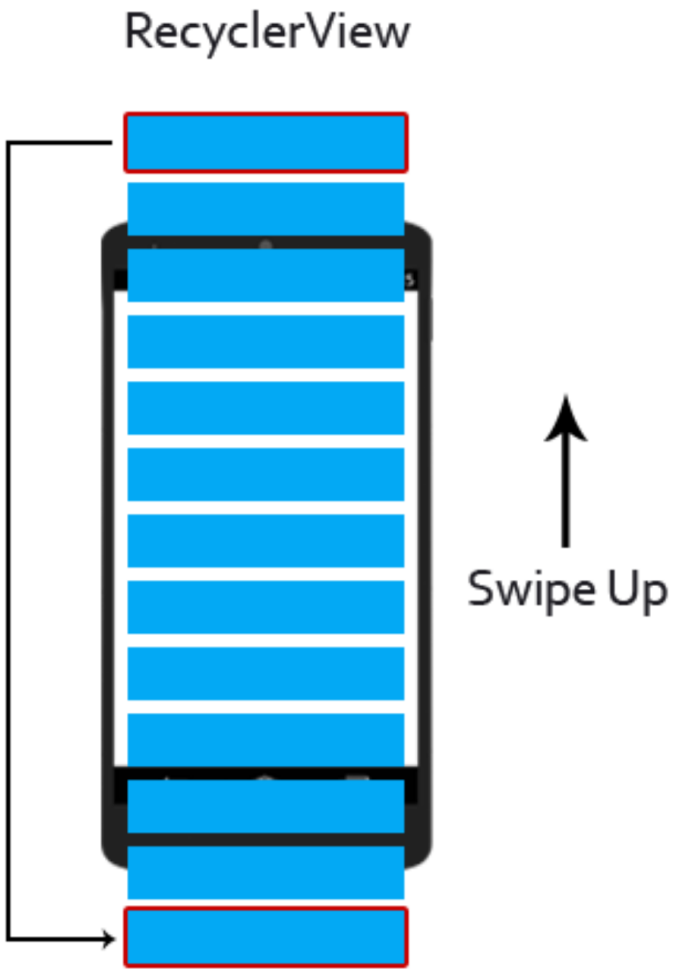
https://developer.android.com/jetpack/compose/lists

Implementing a RecyclerView

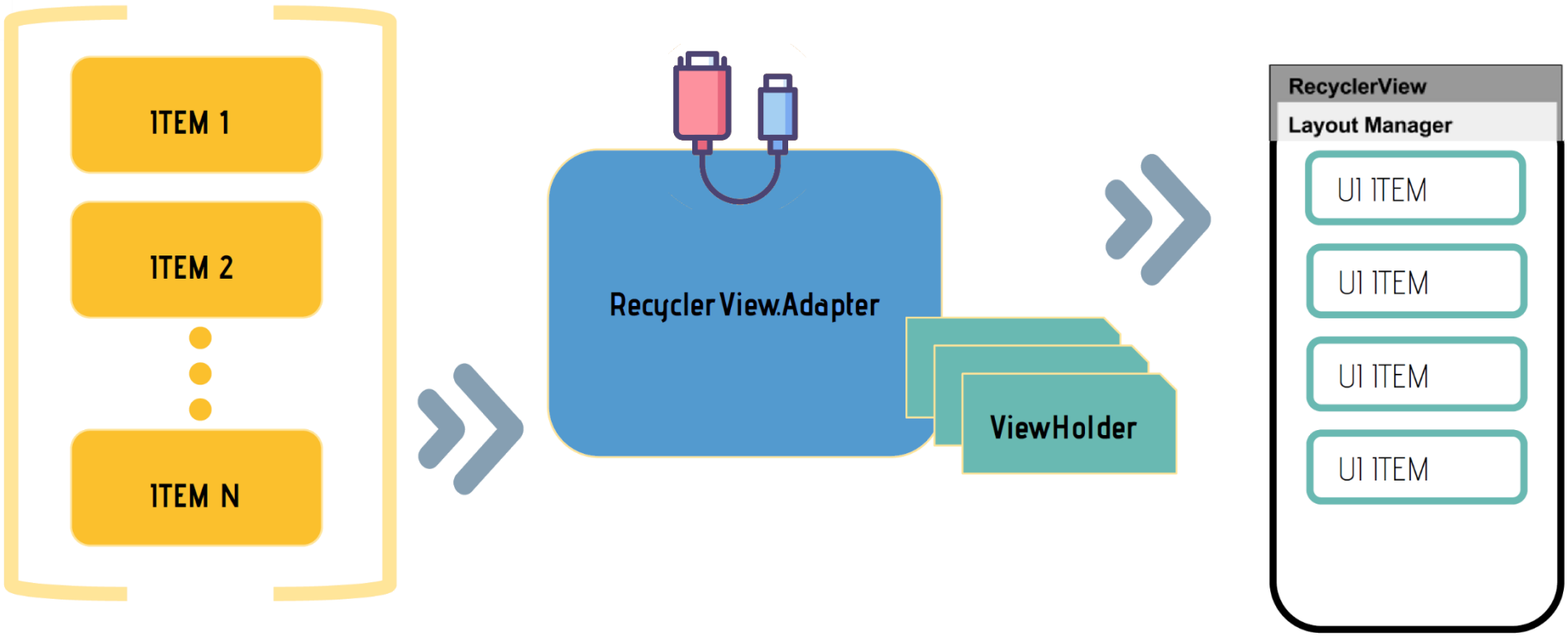
Back

What is a RecyclerView?

- `RecyclerView` is scrollable container for displaying a list of views
- A flexible container for efficiently displaying, and interacting with large sets of data
  - As user scrolls, views that leave the screen are **recycled** and reused to display new items
  - Efficient as it reuses a limited number of views



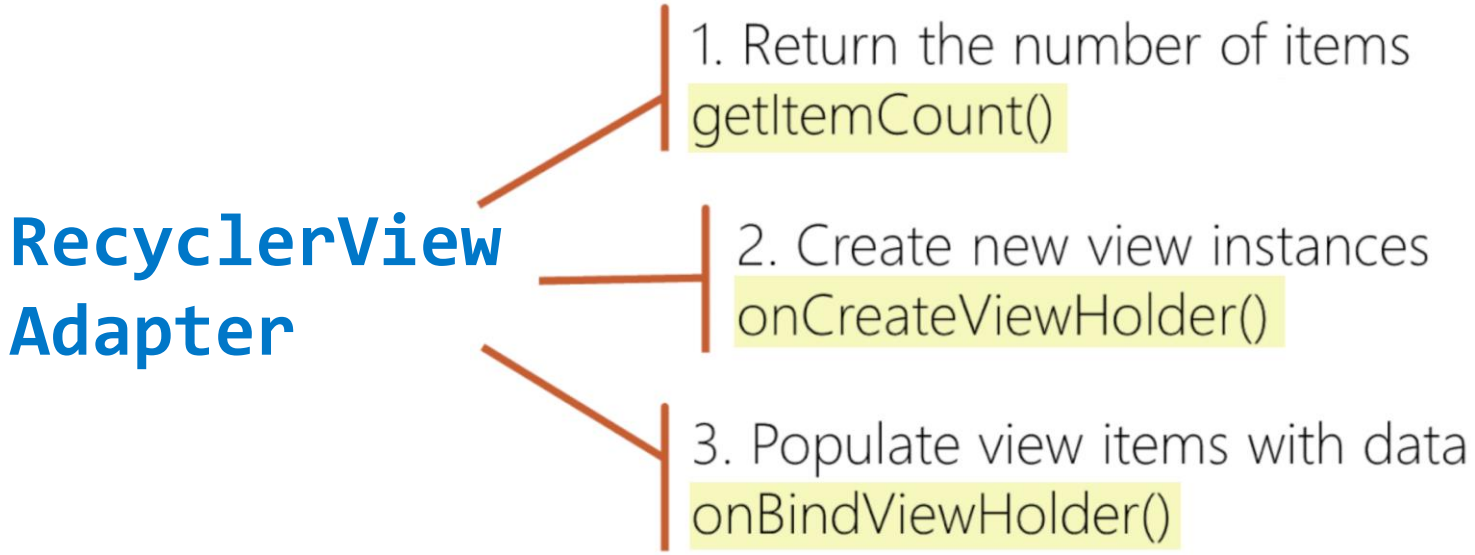
RecyclerView Key Elements



- **Adapter**: connects the *RecyclerView* to the data source
- **ViewHolder:** displays the data in the list item view

What is RecyclerView Adapter?

- In general an adapter helps **incompatible** interfaces work together
- RecyclerView **connects** the RecyclerView to the data source (typically a list of objects):
  - Provides views that represent items in the data source
  - Handles the view logic (such as search, sort)
- It extends RecyclerView.Adapter and implements:

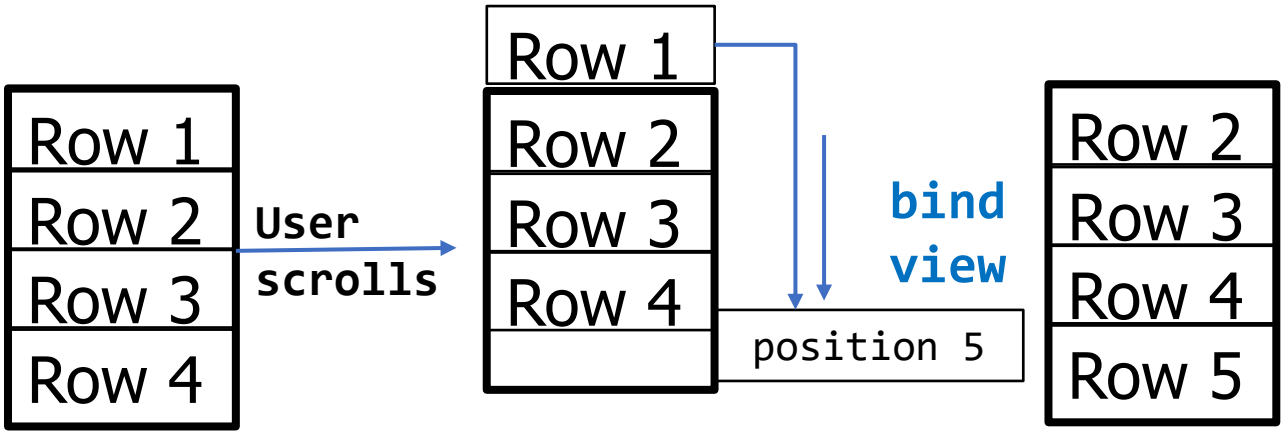


Binding a recycled view

- Initially the RecyclerView calls `onCreateViewHolder` to request the adapter to create a new `ViewHolder` object to hold the inflated item layout

```
val itemView = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent, false)
```

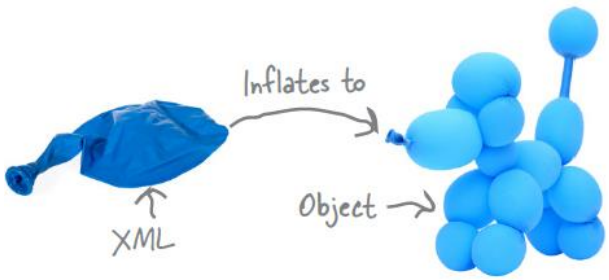
  - Inflation **takes xml layout file and turns it into a View**
  - Inflation is expensive -> RecyclerView minimizes it by reusing views

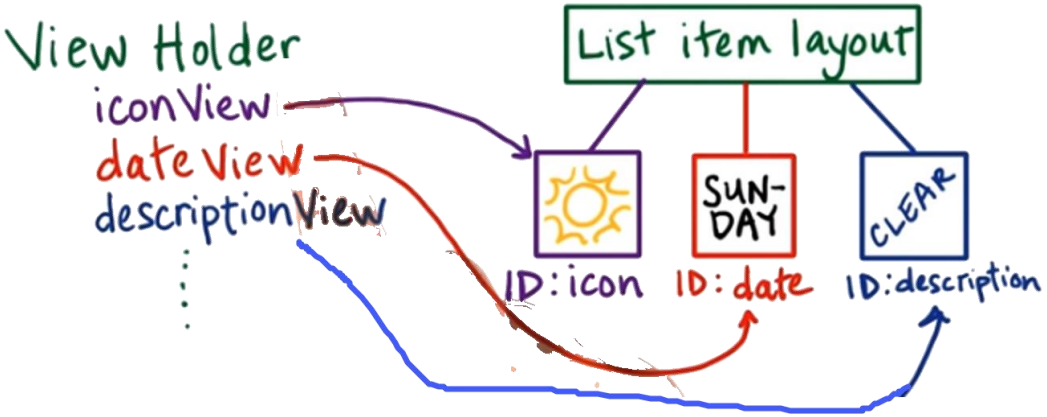



```
val itemView = viewHolder.itemView
val country = countries[position]
itemView.nameTv.text = country.name
itemView.capitalTv.text = country.capital
```

What is a ViewHolder?

- The `ViewHolder` holds the view created from inflating the list item XML Layout





- **onBindViewHolder(viewHolder , position)** the adapter requests the ViewHolder to update the item view using the data object at the requested position (e.g., `countries[position]`)
- Extends [RecyclerView.ViewHolder]

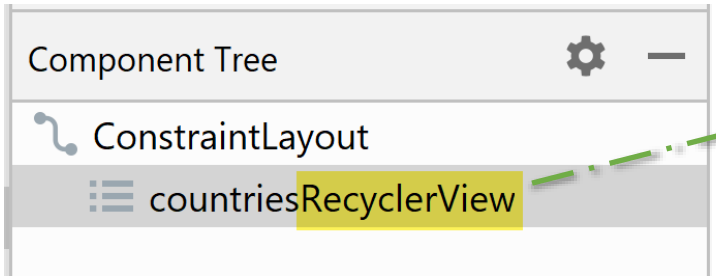
What is a layout manager?

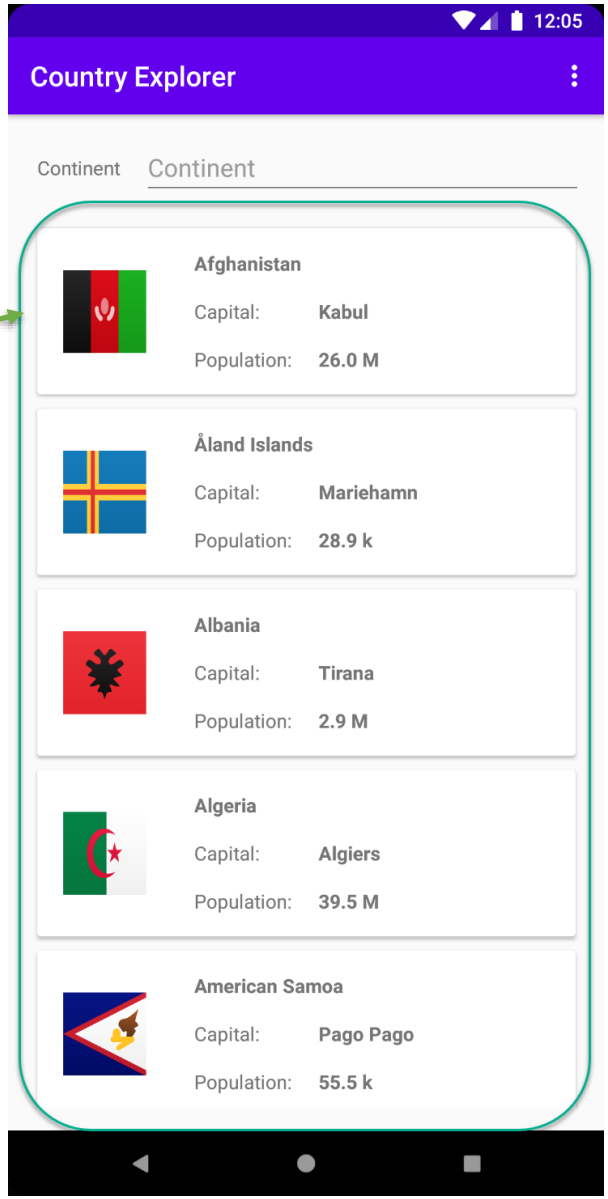
- It **positions** the RecyclerView's items and **tells it when to recycle items** that have transitioned off-screen (i.e., no longer visible to the user)
- Built-in layout managers
  - `LinearLayoutManager:` Lays items out vertically or horizontally
  - `GridLayoutManager:` Lays items out in a grid
  - Others or your own! if you want extra customization

Implementation Steps Summary

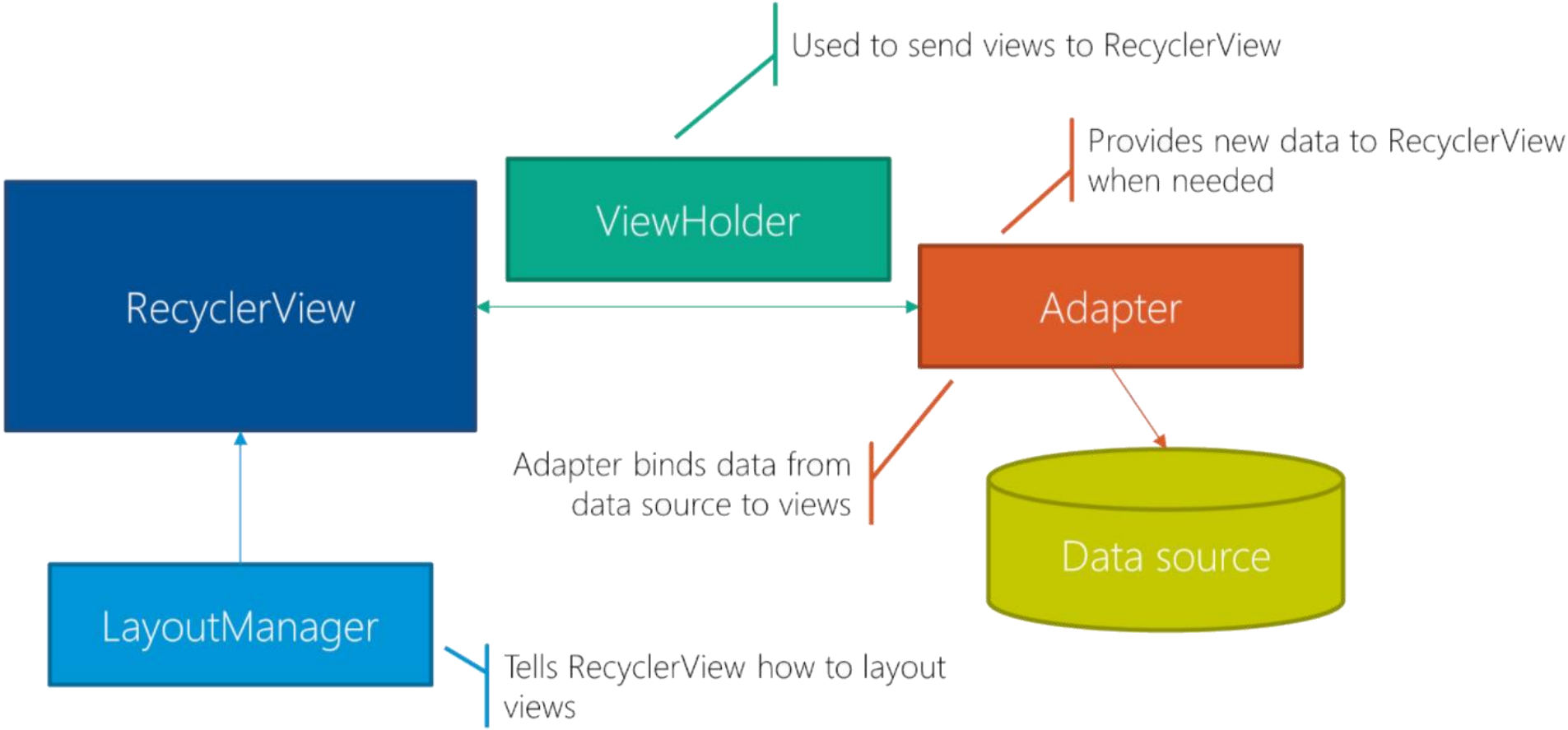
1. Add **`RecyclerView`** to the layout
2. Create **item XML layout** to specify how individual items should be displayed
3. Implement `RecyclerView Adapter` that extends **`RecyclerView.Adapter`**
   - Connect the RecyclerView to the data source and handle the view logic
4. Implement `ViewHolder` that extends **`RecyclerView.ViewHolder`**
   - Holds the inflated view to be used to display individual items
5. In Activity `onCreate`, set the RecyclerView adapter and the `LayoutManager`
   - LayoutManager controls how individual views are laid out on the screen

Country Explorer App



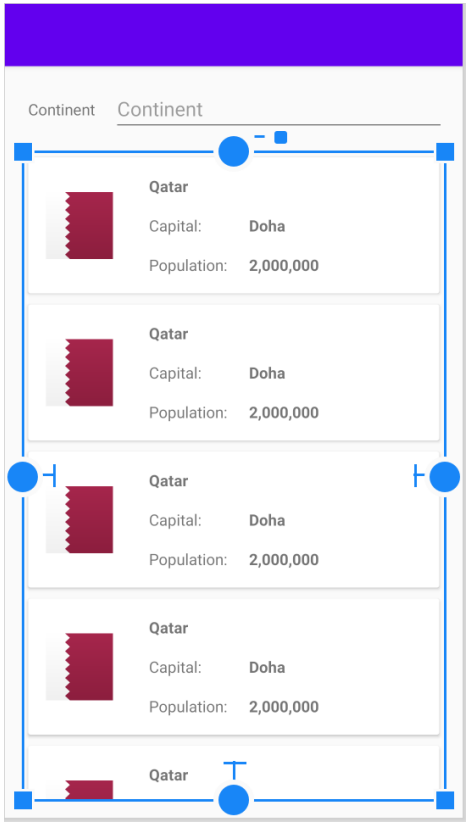


RecyclerView Summary



Tip: Showing List Item Views on the Layout Editor using `tools:listitem`

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/countriesRv"
    tools:listitem="@layout/list_item_country"/>
```



Interacting with RecyclerView

Back

Filter

1. Add **SearchView** to the top toolbar by adding an item to menu_top_toolbar.xml

```
<item
    android:id="@+id/searchMi"
    android:icon="@drawable/ic_search"
    android:title="Search"
    app:actionViewClass="androidx.appcompat.widget.SearchView"
    app:showAsAction="always"
/>
```

2. Handle search by grabbing the searchView from the toolbar and handling the search in Activity onCreate

```
// 2.1 Grab the searchView from the toolbar
val searchView = topToolbar.findViewById<SearchView>(R.id.searchMi)
searchView.setBackgroundColor(Color.WHITE)
// 2.2. Handle search as the user types the search text
searchView.setOnQueryTextListener(searchHandler)
```

3. Write the searchHandler in the activity

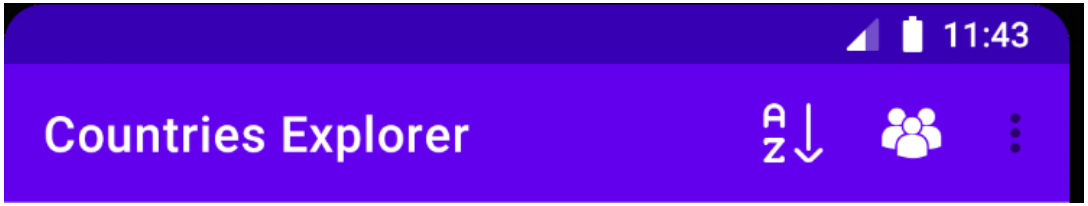
```
private val searchHandler = object : SearchView.OnQueryTextListener {
    // Ignore and do not perform any special behavior here
    override fun onQueryTextSubmit(query: String?) = false

    // As the user types filter the list based on the search text
    override fun onQueryTextChange(searchText: String): Boolean {
        Log.i("CountryListActivity", "Query: $searchText")
        countryAdapter.filter(searchText)
        return true
    }
}
```

4. Filter the data list in the RecyclerView Adapter

```
fun filter(searchText: String) {
    countryFilteredList = if (searchText.isEmpty()) {
        countries
    } else {
        countries.filter { it.continent.contains(searchText, true) or
                           it.name.contains(searchText, true) or
                           it.capital.contains(searchText, true) or
                           it.code.contains(searchText, true)
        }.toMutableList()
    }
    notifyDataSetChanged()
}
```

Sort

1. First add a toolbar to the activity layout




```
<com.google.android.material.appbar.AppBarLayout
    android:id="@+id/top_app_bar">

  <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/topToolbar"
        app:menu="@menu/menu_top_toolbar"
        app:title="@string/countries_list_title" />

</com.google.android.material.appbar.AppBarLayout>
```

2. Define of the toolbar menu items in menu_top_toolbar.xml

```
<menu  ... >
    <!-- Setting app:showAsAction="never" will show the menu
     option under Overflow menu accessible from ellipses ... -->
    <item android:id="@+id/sortByNameMi"
        android:title="Sort by name"
        android:icon="@drawable/ic_sort"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/sortByPopulationMi"
        android:title="Sort by population"
        android:icon="@drawable/ic_population"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/sortByPopulationDescendingMi"
        android:title="Sort by population - Descending"
        app:showAsAction="never" />
</menu>
```

3. Handle toolbar menu item clicked

- Add `topToolbar.setOnMenuItemClickListener` to the activity onCreate method

```
topToolbar.setOnMenuItemClickListener { onToolbarMenuItemClicked(it) }

private fun onToolbarMenuItemClicked(menuItem: MenuItem): Boolean {
    val sortBy = when (menuItem.itemId) {
        R.id.sortByNameMi -> SortBy.NAME
        R.id.sortByPopulationMi -> SortBy.POPULATION
        R.id.sortByPopulationDescendingMi -> SortBy.POPULATION_DESC
        else -> null
    }
    countryAdapter.sort(sortBy!!)
    return true
}
```

4. RecyclerViewAdapter sorts the list

```
fun sort(sortBy: SortBy) {
  countries = when (sortBy) {
    SortBy.NAME -> countries.sortedBy { it.name }
    SortBy.POPULATION ->
          countries.sortedBy { it.population }
    SortBy.POPULATION_DESC ->
           countries.sortedByDescending { it.population }
  }
  // Caused a refresh of the RecyclerView
  notifyDataSetChanged()
}
```

Handling Item Clicked Event

1. Add a parameter to the Adapter to receive the **`clickListener`**

```
class CountryAdapter(private val countries: List<Country>,
                     private val clickListener: (Country) -> Unit)
    : RecyclerView.Adapter<RecyclerView.ViewHolder>()
```

2. Add the **`clickListener`** to every List Item View

```
inner class CountryViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    fun bind(country: Country) {  ...
        itemView.setOnClickListener { clickListener(country) }
    }
}
```

3. In the Activity create a **`clickListener`** and pass it to the Adapter

```
countriesRv.adapter = CountryAdapter(countries, ::onCountryClicked)
...
private fun onCountryClicked(country : Country) {
    toast("Clicked: ${country.name}", Toast.LENGTH_LONG)
}
```

Handling Item Clicked Event

- In its constructor, the *Adapter* requires a reference to the data source and a **click handler**
  - Whenever the adapter is instructed by the *RecyclerView* to bind a new *ViewHolder*, the bind method **attaches the click listener** to the item view

```
itemView.setOnClickListener { clickListener(country) }
```

  - This allows individual items in the list to handle click events
- When the user later clicks on an item in the *RecyclerView*, the click listener is executed
  - As parameter, the listener gets the data list element (e.g., country object) that was clicked, to react accordingly

1. Swipe to Delete and Undo

```
/* 1. Create ItemTouchHelper.SimpleCallback and tell it what events to listen for.
It takes two parameters: One for drag directions and one for swipe directions.
We're only interested in swipe. Pass 0 to inform the callback not to respond to drag events. */
val swipeHandler = object :
    ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.LEFT or ItemTouchHelper.RIGHT) {
    // Ignore and do not perform any special behavior here
    override fun onMove( ... ) = false

    // 2. onSwiped ask the RecyclerView adapter to delete the swiped item
    override fun onSwiped(viewHolder: RecyclerView.ViewHolder, swipeDirection: Int) {
        countryAdapter.deleteCountry(viewHolder)
    }
}

/* 3. Initialize ItemTouchHelper with the swipeHandler you defined,
      and then attach it to the RecyclerView. */
val itemTouchHelper = ItemTouchHelper(swipeHandler)
itemTouchHelper.attachToRecyclerView(countriesRv)
```

2. RecyclerViewAdapter deletes the swiped item

```
fun deleteCountry(viewHolder: RecyclerView.ViewHolder) {
    // Get the position of the item that was swiped
    val position = viewHolder.adapterPosition
    deletedCountry = countries[position]
    countries.removeAt(position)

    // Inform the RecyclerView adapter that an item has been
    // removed at a specific position.
    notifyItemRemoved(position)

    Snackbar.make(viewHolder.itemView, "${deletedCountry.name}
        removed", Snackbar.LENGTH_LONG).setAction("UNDO") {
            countries.add(position, deletedCountry)
            notifyItemInserted(position)
    }.show()
}
```

Greenland removed UNDO

Resources

- RecyclerView codelab
  - https://codelabs.developers.google.com/codelabs/kotlin-android-training-recyclerview-fundamentals/

- Handling RecyclerView click event
  - https://www.andreasjakl.com/recyclerview-kotlin-style-click-listener-android/